Shell Programming (Contd…)

Lecture : 04

Animesh Dutta (Sessional in-charge)

Sessional assistants:

Abhisek Roy

Narayan Chongder

Sangeeta Sen (Das)

Amrita Namtirtha

Samriddhi Sarkar

Rima Hazra

Pankaj Vishwakarma

Vivek Verma

Topics to be covered

1. Read only and Unsetting of variable.
2. Special variable in Shell.
3. Conditional statement.
4. Looping.
5. Array.
6. Function.
7. Shell substitution.
8. Output and Input redirection.
9. Operator in Shell.

Read only Variables

The shell provides a way to mark variables as **read-only** by using the **read only command**. After a variable is **marked read-only**, its **value cannot be changed**.

Example:

```
#!/bin/sh
NAME="Eric Bensana"
readonly NAME
NAME="Lametire"
```

Output:

```
/bin/sh: NAME: This variable is read only
```

Unsetting or deleting Variables

- **Unsetting or deleting** a **variable** is to **remove** the **variable** from the **list of variables** that it **shell** tracks.

- Once we unset a variable, we would not be able to access stored value in the variable.

Example:

```
#!/bin/sh
NAME="Zara Ali"
unset NAME
echo $NAME
```

Output:

Above example would not print anything.

Note: We **cannot use** the **unset command** to unset variables that are marked **read only**.

Special Variables

Variable	Description
$0	Shows the file name of the current script.
$n	It correspond to the command line arguments. For example: the first argument is $1, the second argument is $2, and so on.
$* and $@	● Provides access all of the command-line arguments at once. ● They will act as same unless they are enclosed in double quotes, "".
“$*” and “$@”	"$*" takes the entire list as one argument with spaces between. "$@" takes the entire list and separates it into separate arguments.
$?	The exit status of the last executed command .
$$	The process ID under which shell scripts are executing of the current shell.

Examples on Special Variables

Program:

test.sh

```
#!/bin/sh
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo  $@
echo  $*
echo "Total Number of Parameters : $#"
for TOKEN in $*
do
   echo $TOKEN
Done
echo $?
```

Run: ./test.sh Eric Bensana 60 Years Old

Output:

```
File Name: ./test.sh
First Parameter : Eric
Second Parameter : Bensana
Eric Bensana 60 Years Old
Eric Bensana 60 Years Old
Total Number of Parameters : 5
Eric
Bensana
60
Years
Old
0
```

Examples on Special Variables

Program:

```
#!/bin/sh
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo  $@
echo  $*
echo "Total Number of Parameters : $#"
for TOKEN in “$*”
do
   echo $TOKEN
Done
Echo $?                  # Exit Status
```

Run: ./test.sh Eric Bensana 60 Years Old

Output:

File Name: ./test.sh

First Parameter : Eric

Second Parameter : Bensana

Eric Bensana 60 Years Old

Eric Bensana 60 Years Old

Total Number of Parameters : 5

Eric Bensana 60 Years Old

0

Exit status is a numerical value returned by every command upon its completion.

As a rule, most commands return an exit status of **0** (success), and **1** (not success**)**.

Conditional Statement

```
if [ expression ]
then
   Statement(s) to be
   executed if expression is
   true
fi
```

Output: a is not equal to b

Example:

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
   echo "a is equal to b"
fi
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

Conditional Statement (Contd...)

```
if [ expression ]
then
   Statement(s) to be executed
    if expression is true
else
   Statement(s) to be executed
    if expression is not true
fi
```

Example:

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
   echo "a is equal to b"
else
   echo "a is not equal to b"
fi
```

Output: a is not equal to b

Conditional Statement (Contd...)

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression
     1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression
     2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression
     3 is true
else
   Statement(s) to be executed if no
     expression is true
fi
```

Example:

```
a=10
b=20
if [ $a == $b ]  then
echo "a is equal to b"
elif [ $a -gt $b ]
then
echo "a is greater than b"
elif [ $a -lt $b ]
then
echo "a is less than b"
else
echo "None of the condition met"
fi
```

Output: a is less than b

Conditional Statement (Contd...)

```
case word in

pattern1)

    Statement(s) to be executed if
    pattern1 matches ;;

  pattern2)

    Statement(s) to be executed if
    pattern2 matches ;;

  pattern3)

    Statement(s) to be executed if
    pattern3 matches ;;

esac
```

Output:

```
$ ./test2.sh -f ipe.eps

File name is ipe.eps

$ ./test2.sh -* usr
```

```
#!/bin/sh

option="${1}"

case ${option} in

  -f) FILE="$2"

    echo "File name is $FILE“  ;;

  -d) DIR="$2"

    echo "Dir name is $DIR" ;;

  *)

    echo "`basename $0`:usage: [-f
    file] | [-d directory]"

    exit 1 # Command to come out of
    the program with status 1 ;;

  esac
```

```
test2.sh:usage: [-f file] | [-d directory]

$ ./test2.sh -d usr

Dir name is usr
```

Looping

```
while <command>
do
   Statement(s) to be executed if
    command is true
Done
```

Example:

```
#!/bin/sh
a=0
while [ $a -lt 9 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Output:

```
0
1
2
3
4
5
6
7
8
```

Looping (Contd...)

```
for var in word1 word2 ... wordN
do
   Statement(s) to be executed for
    every word.
done
```

Example:

```
#!/bin/sh
for var in 0 1 2 3 4 5 6 7 8 9
do
   echo $var
done
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

Looping (Contd...)

until command

do

  Statement(s) to be executed until command is true

done

Example:

```
#!/bin/sh
a=0
until [ ! $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

Looping (Contd...)

```
select var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

Output:

```
$ ./test3.sh
1) Option 1
2) Option 2
3) Option 3
4) Quit
Please enter your choice: 1
you chose choice 1
Please enter your choice: 2
you chose choice 2
Please enter your choice: 4
$
```

Example:

```
PS3='Please enter your choice: '
options=("Option 1" "Option 2" "Option 3" "Quit")
select opt in "${options[@]}"
do
    case $opt in
      "Option 1") echo "you chose choice 1" ;;
       "Option 2") echo "you chose choice 2" ;;
       "Option 3") echo "you chose choice 3" ;;
       "Quit") break ;;
       *) echo invalid option;;
    esac
done
done
```

Array

Defining Array Values:

array_name[index]=value

Accessing Array Values:

${array_name[index]}

Output:

$./test.sh

First Index: Zara

Second Index: Qadir

First Method: Zara Qadir Mahnaz Ayan Daisy

Second Method: Zara Qadir Mahnaz Ayan Daisy

Example:

```
#!/bin/sh

NAME[0]="Zara"

NAME[1]="Qadir"

NAME[2]="Mahnaz"

NAME[3]="Ayan"

NAME[4]="Daisy"

echo "First Index: ${NAME[0]}"

echo "Second Index: $
    {NAME[1]}"

echo "First Method: ${NAME[*]}"

echo "Second Method: $
    {NAME[@]}"
```

Function

Creating Functions:

```
function_name () {
   list of commands
}
```

Output1:

```
$./test.sh
Hello World
```

Output2:

```
$./test.sh
Hello World Zara Ali
```

Example1:

```
#!/bin/sh
# Define your function here
Hello () {
   echo "Hello World“
# Invoke your function
Hello
```

Example2: (parameter passing)

```
#!/bin/sh
# Define your function here
Hello () {
   echo "Hello World $1 $2"
}# Invoke your function
Hello Zara Ali
```

Function (Contd...)

Returning Values from Functions:

return val

Output:

$./test.sh

Hello World Zara Ali

Return value is 10

Example:

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World $1 $2"
   return 10
}

# Invoke your function
Hello Zara Ali

# Capture value returned by last
   command
ret=$?

echo "Return value is $ret"
```

Function (Contd...)

Nested Functions

Example:

```
#!/bin/sh

# Calling one function from another

number_one () {

   echo "This is the first function
    speaking..."

   number_two

}

number_two () {

   echo "This is now the second
    function speaking..."

}

# Calling function number_one.

number_one
```

Output:

This is the first function speaking...

This is now the second function speaking...

Function Removal:

$unset .f function_name

Shell Substitution

Escape sequence substitution:

```
#!/bin/sh
a=10
echo -e “\t Value of a is $a "
```

Output:

Value of a is 10

Note:

-e option enables interpretation of backslash escapes..

-n option to disable insertion of new line.

This is true for other backslash escapes (for e.g \b, \f, \v, \a, etc.)

Command substitution by variable:

```
#!/bin/sh
DATE=`date`
echo "Date is $DATE"
USERS=`who | wc -l`
echo "Logged in user are $USERS“
```

Output:

Date is Thu Jul 2 03:59:57 MST 2009

Logged in user are 1

Output and input redirection

Output redirection:

- The output from a command diverted to a file instead of standard output (STDOUT) .
- $ command > file
- File will automatically be created @ present directory.

Example:

```
#!/bin/sh
who > user        #  user is a file.
date >> user      #  >> for append.
pwd >> user
```

Output: (content of user)

```
user     :0          2016-02-08 01:53 (:0)
user     pts/0       2016-02-08 01:54
    (:0)
Mon Feb  8 03:04:28 IST 2016
/home/user
```

Output and input redirection (Contd…)

1. Input redirection:

- Input of a command is redirected from a file.
- $ command < file

Output 1:

$ ./test.sh

4

17

2. End Of File (EOF):

- << operator is used to instruct read operation to continue from input until EOF delimiter is reached.
- command << delimiter

Example 1:

```
#!/bin/sh
wc -l < user
#word_count = wc -w < user
wc -w < user
```

Example2:

```
cat user << EOF
```

Output2: content of file user.

Example3:

```
wc -l << EOF
```

Output3: 4

Operators in shells scripts

Arithmetic Operators

Operator	Example
+	`expr $a + $b` will give 30
-	`expr $a - $b` will give -10
*	`expr $a \* $b` will give 200
/	`expr $b / $a` will give 2
%	`expr $b % $a` will give 0
=	a=$b would assign value of b into a
==	[$a == $b] would return false.
!=	[$a != $b] would return true.

For example variable a holds 10 and variable b holds 20.

Logical Operators

Operator	Example
!	[! false] is true.
-o	[$a -lt 20 -o $b -gt 100] is true.
-a	[$a -lt 20 -a $b -gt 100] is false.

For example variable a holds 10 and variable b holds 20.

String Operators

Operator	Example
= (String Comparison)	[$a = $b] is not true.
!= (- do -)	[$a != $b] is true.
-z (If it is zero length then it returns true.)	[-z $a] is not true.
-n (If it is non-zero length then it returns true.)	[-n $a] is not false.
str (If it is empty then it returns false.)	[$a] is not false.

For example variable a holds "abc" and variable b holds "efg".

Thank You

Assignments

1. Write Shell scripts to implement Bubble, Selection, Merge and Quick sort techniques.
2. Write a Shell script to implement built in functions for String handling in C. For example strlen(), strcat(), strcmp() and strrev().